Jinja2
Templating engine za Python

Sadržaj

Uvod

- Jinja2 je moderni template engine za Python
- Modelovan prema Django templatima
- Pruža HTML escaping sistem za prevenciju XSS-a
- Omogućava nasleđivanje template-a
- Template engine opšte namene

Instalacija

- Koristeći *easy-install*
  - easy_install Jinja2
- Koristeći *pip*
  - pip install Jinja2
- Jinja2 radi sa Python-om 2.6 i 2.7 kao i Python-om >= 3.3
- Novije verzije ne podržavaju Python 3.2 (skinuti verziju Jinja-e stariju od 2.7 ako nam treba podrška za starije verzije Python-a 3)

Template (šablon)

- Template je običan tekstualni fajl. Može generisati bilo koji text-based format, ne samo html ili xml.
- Može sadržati varijable i izraze koje se zamenjuju konkretnim vrednostima kada se template evaluira
- Sintaksa je inspirisana Python-om i Djangom

Izrazi (1)

- Jinja, poput Python-a dozvoljava da se osnovni izrazi pojave na bilo kom mestu
- Najjednostavniji oblik izraza su literali , koji su zapravo reprezentacije Python objekata poput stringova i brojeva
  - na raspolaganju su stringovi, brojevi, liste, torke, rečnici i boolean true i false vrednosti
  - isti kao i u Python-u (recimo, liste se pišu unutar [...], torke unutar (...) i sl.
  - jedina razlika je što se true i false pišu malim slovom (mada u novijim verzijama može i velikim, ali se ne preporučuje)

Izrazi (2)

- Jinja podržava i matematičke operatore, mada je to retko od koristi u template-ima:
  - računanje (+, -, /, //, %, *, **)
  - poređenje (==, !=, >, >=, <, <=)
  - logičke (and, or i not)
- Preostali operatori, koji se ne mogu svrstati ni u jednu kategoriju, ali su veoma korisni
  - in (proverava da li neki element pripada nekoj sekvenci)
  - is (vrši test, više u nastavku)

Izrazi (3)

- | (primenjuje filter, više u nastavku)
- ( ) (poziva callable)
- . ili [ ] (vraća atribut objekta)
- ~ (konvertuje sve operande u stringove i konkatanira ih)

- Na kraju imamo i if izraz, koji se može koristiti unutar tagova, inline, na primer:

```
{% extends layout_template if layout_template is
defined else 'master.html' %}
```

Varijable (1)

- Varijable koje se proslede iz aplikacije se mogu koristiti u template-ima
- Ako sadrže atribute ili elemente (liste, rečnici, skupovi), može se pristupati i njima
  - atributima se može pristupati na klasičan način, tipa foo.bar
  - ali i pomoću takozvane subscript sintakse: foo['bar']
- Ako varijabla ili atribut ne postoji, po defaultu se evaluira na prazan string (može se podesiti i drugačije ponašanje)

Varijable (2)

- Sintaksa foo.bar rezultuje sledećim
  - proverava se da li postoji atribut *bar* u *foo*
  - ako ne postoji, proverava da li postoji element 'bar' u *foo*
  - ako ni to ne postoji, vraća undefined objekat
- Sintaksa foo['bar'] rezultuje sledećim
  - proverava da li postoji element 'bar' u *foo*
  - ako ne postoji, *p*roverava se da li postoji atribut *bar* u *foo*
  - ako ni to ne postoji, vraća undefined objekat

Ispis vrednosti izraza

- Rezultat izraza se ispisuje po renderovanju ukoliko se izraz stavi unutar {{ ... }}
- Recimo:

```
{{ var }}
```

ispisuje vrednost varijable var

Komentari

- Da bi se zakomentarisao neki deo template-a, koristi se sintaksa {# ... #}
- Komentar može da se prostire kroz više redova template-a

Whitespace kontrola

- Po defaultu, template engine whitespace karaktere vraća nepromenjene
- Može se konfigurisati sledeće:
  - trim_blocks - prvi prelazak u novi red nakon tagova se uklanja
  - lstrip_blocks - uklanjaju se svi tabovi i razmaci između početka linije i početka bloka
  - lstrip_blocks se može i uključiti ako se u template-u na početak bloka stavi +, a isključiti ako se stavi -. Recimo:

```
{%+ if something %} yay {% endif %}
```

Escaping

- Ako je potrebno da delovi koje bi bili prepoznati kao varijable ili blokove budu prikazani takvi kakvi jesu, koristi se tag *raw*.

```
{% raw %}
    <ul>
    {% for item in seq %}
        <li>{{ item }}</li>
    {% endfor %}
    </ul>
{% endraw %}
```

Kontrolne strukture

- Pod kontrolnom strukturom smatra se sve što kontroliše tok, poput kondicionala, for petlji, ali i makroa i blokova
- Kontrolna struktura se navodi unutar {% ... %} bloka

For

- Iteriranje kroz neku sekvencu
- Sintaksa inspirisana for petljom u Python-u

```
{% for user in users %}
    ...
{% endfor %}
```

- ili

```
{% for key, value in dictionary.items() %}
    ...
{% endfor %}
```

Specijalne varijable unutar for petlje

- Unutar for petlje može se pristupati nekolicini specijalnih varijabli
  - loop.index - tekuća iteracija (sa početnim indeksom 1)
  - loop.index0 - tekuća iteracija (sa početnim indeksom 0)
  - loop.revindex i loop.revindex0 - broj iteracija od kraja petlja (1 indeksirano i 0 indeksirano)
  - loop.first - True ako se radi o prvoj iteraciji
  - loop.last - True ako se radi o poslednjoj iteraciji
  - loop.length - Broj elemenata u sekvenci

Cycle

- Unutar for petlje može se ciklično iterirati kroz listu stringova ili varijabli pomoću specijanle varijable loop.cycle

```
{% for row in rows %}
    <li class="{{ loop.cycle('odd','even') }}">
        {{ row }}
     </li>
{% endfor %}
```

Filtriranje sekvence

- Ne postoji break ili continue naredba, kao u Python-u, ali se zato mogu preskočiti neki elementi sekvence

```
{% for user in users if not user.hidden %}
    <li>{{ user.username }}</li>
{% endfor %}
```

- Specijalne varijable neće brojati preskočene elemente

For - else

- Ako je sekvena kroz koju se treba iterirati prazna, ili su svi njeni članovi uklonjeni filtriranjem, može se staviti else blok u koji se ulazi samo ako je ovaj uslov zadovoljen:

```
{% for user in users %}
    <li>{{ user.username}}</li>
{% else %}
    <li><em>Nema nijednog korisnika</em></li>
{% endfor %}
```

If

- Veoma nalik na if u Python-u, može imati više elif grana

```
{% if kenny.sick %}
    Kenny is sick.
{% elif kenny.dead %}
    You killed Kenny! You bastard!!!
{% else %}
    Kenny looks okay --- so far
{% endif %}
```

Makroi (1)

- Makroi su poput funkcija programskih jezika gde se može izdvojiti nšto što bi se inače više puta ponavljalo
- Recimo,

```
{% macro input(name, value='', type='text', size=20)%}
    <input type="{{ type }}" name="{{ name }}"
value="{{ value }}" size="{{ size }}">
{% endmacro %}
```

- Ako naziv makroa počinje sa _, dati makro se neće importovati u druge template

Makroi (2)

- Poput argumenata funkcija u Python-u i argumenti makroa mogu imati defaultne vrednosti
- Takođe, makro može imati promenljiv broj argumenata, opet slično kao i u Python-u
  - ovi argumenti se smeštaju u specijalne promenljive varargs i kwargs
- Definisani makro se može pozvati slično kao funkcija

```
<p>{{ input('username') }}</p>
<p>{{ input('password', type='password') }}</p>
```

Call blok (1)

- Nekada je korisno jedan makro proslediti drugom, za šta se može koristiti call blok
- Call blok fukcioniše isto kao makro, ali nema ime
- Ako se makro pozove is call taga, specijalna varijabla tog makroa, *caller* će čuvati informaciju o tome ko je pozvao makro

Call blok (2)

```
{% macro render_dialog(title, class='dialog') %}
        <h2>{{ title }}</h2>
        <div class="contents">
            {{ caller() }}
        </div>
{% endmacro %}

{% call render_dialog('Hello World') %}
    This is a simple dialog rendered by using a macro and
    a call block.
{% endcall %}
```

Call blok (3)

- Ispis bi tada bio sledeći:

```
  <h2>Hello World</h2>
    <div class="contents">
     This is a simple dialog rendered by using a
macro and a call block.
    </div>
```

- Moguće je i proslediti argumente call bloku:

```
{% call(argument) render_dialog('Hello World') %}
    {{argument}}
{% endcall %}
```

Dodela vrednosti (1)

- Dodele vrednosti mogu biti unutar blokova, makroa ili petlji, ali i van njih, kada se importuju u druge template
- Koristi se *set* tag

```
{% set navigation = [('index.html', 'Index'),
('about.html', 'About')] %}

{% set key, value = call_something() %}
```

Dodela vrednosti (2)

- Od verzije Jinja 2.8 sadržaj jednog bloka može se povezati sa varijablom
- Ovo se može koristiti kao alternativa makroima u nekim situacijama
- Sve što se nalazi između {% set var %} i {% endset %} se čuva u varijabli var

```
{% set navigation %}
    <li><a href="/">Index</a>
    <li><a href="/downloads">Downloads</a>
{% endset %}
```

Nasleđivanje

- Omogućava definisanje osnovnog template-a koji sadrži zajedničke elemente više drugih, kao i blokove koje naslednici mogu redefinisati
- Blok tag samo govori template engine-u da naslednički template može redefinisati te delove template-a
- Extends tag, sa druge strane, govori template engine-u da dati template nasleđuje drugi. Mora biti prvi tag u template-u

Base template (1)

```
<html>
<head>
    {% block head %}
    <link rel="stylesheet" href="style.css" />
    <title>{% block title %}{% endblock %}</title>
    {% endblock %}
</head>
```

Base template (2)

```
<body>
    <div id="content">{% block content %}{% endblock %}</div>
    <div id="footer">
        {% block footer %}
        &copy; Copyright 2014 by <a href="http://domain/">you</a>.
        {% endblock %}
    </div>
</body
</html>
```

Naslednički template (1)

```
{% extends "base.html" %}
{% block title %}Index{% endblock %}
{% block head %}
    {{ super() }}
    <style type="text/css">
        .important { color: #336699; }
    </style>
{% endblock %}
```

Naslednički template (2)

```
{% block content %}
    <h1>Index</h1>
    <p class="important">
      Welcome on my awesome homepage.
    </p>
{% endblock %}
```

- Ako naslednički template ne definiše neki blok (u ovom slučaju footer), prikazaće se sadržaj roditeljskog
- Ne može se definisati više blokova istog imena u jednom template-u

Super blok i imenovani endblok tagovi

- Super blok omogućava renderovanje sadržaja roditljskog bloka

```
{% block sidebar %}
    <h3>Table Of Contents</h3>
    ...
    {{ super() }}
{% endblock sidebar %}
```

- Radi dobijanja preglednijih template-a, u endblock tagu može se staviti naziv bloka koji se završava

Include (1)

- Include se koristi kada želimo da uključimo neki template i njegov renderovani sadržaj prikažemo u tekućem template-u
- Template-i koji se referenciraju na ovaj način po defaultu imaju pristup varijablama aktivnog konteksta (što znači da može koristiti varijable poznate tekućem template-u)

```
{% include 'header.html' %}

    Body

{% include 'footer.html' %}
```

Include (2)

- Od verzije Jinja 2.2 include se može kombinovati sa *ignore missing*, čime se obezbeđuje da ako referencirani template ne postoji, neće biti bačena greška, nego jednostavno ništa na tom mestu neće biti prikazano
- Može se navesti i više template-a, kada se redom od prvog do poslednjeg proverava da li postoje, pa se prvi koji postoji uključuje

```
{% include ['special_sidebar.html', 'sidebar.html'] ignore missing %}
```

Import (1)

- Često se makroi definišu u posebnom template-u pa onda importuju
- Import funkcioniše slično kao Python-ov
- Po defaultu, importovani template ne može prisupati varijablama template-a koji ga importuje (za razliku od include-a)
- Može se importovati čitav template ili samo neki njegovi konkretni delovi

Import (2)

- Recimo da imao template forms.html i u njemu makroe input i textarea
- Kao i u Python-u imamo dva načina importovanja:

1)
```
{% import 'forms.html' as forms %}
{{ forms.input('username') }}
```

2)
```
{% from 'forms.html' import input as input_field,
textarea %}

{{ input_field('username') }}
```

Filteri

- Varijable se mogu modifikovati upotrebon filtera
- Filteri se od varijable odvajaju upotrebon | operatora i opciono mogu imati argumente koji se navode u zagradama
- Filteri se mogu ulančavati
- Primer:

```
{{ list|join(', ') }}
```

- Lista ugrađenih filtera se može naći na linku

Testovi (1)

- Osim filtera na raspolaganju su i testovi, koji se mogu koristiti za proveru da li neka varijabla ili izraz zadovoljavaju određeni kriterijum
- Opšta sintaksa je: variable *is* test
- Test može da prima argumente, koji se onda navode u okviru zagrada
- Ako test prima samo jedan argument, on se može napisati i bez zagrada

Testovi (2)

- Primer:

```
{% if loop.index is divisibleby 3 %}
{% if loop.index is divisibleby(3) %}
```

- Lista ugrađenih testova može se naći na linku
- Najinteresantniji je defined(variable), koji vraća true ako je varijabla definisana:

```
{% if variable is defined %}
    value of variable: {{ variable }}
{% else %} ...
```

Globalne funkcije (1)

- Sledeće funckije se, po defaultu, mogu koristiti u svakom template-u
  - range([start,]stop[, step]) - praktično ista kao i Python-ova istoimena funkcije
  - lipsum(n=5, html=True, min=20, max=100) - generiše n paragrafa dužine između min i max, običnog ili html teksta, pogodno za testiranje
  - dict(**items) - alternativa dict literalima {'foo': 'bar'} je isto što i dict('foo'='bar').

Globalne funkcije (2)

- cycler(*items) - omogućava kretanje kroz listu vrednosti, slično kao loop.cycle, samo se može koristiti i van petlji i u više petlja
  - reset() - resetuje cycle na prvi element
  - next() - ide na sledeći element u ciklusu i tada vrati trenutni
  - current() - vraća trenutni elemet
- joiner - koristi se za spajanje više sekcija

- Detaljnije na linku

Ekstenzije (1)

- Jinja sadrži nekoliko ugrađenih ekstenzija, koje se mogu omogućiti u aplikacijama
  - i18n - omogućava prevođenje delova template-a, koristeći {% trans %} tag
  - expression statement - omogućava upotrebu taga *do,* koji radi isto što i {{...}} samo bez ispisivanja ičega (recimo, unutar njega možemo dodati element u listu i sl.)

Ekstenzije (2)

- loop control - omogućava upotrebu break i continue naredbi u petljama ({% break %} i {% continue %})
- with statemtnt - omogućava upotrebu *with* taga, koji definiše unutrašnji scope, tako da su sve promenljive između {% with %} i {% endwith %} lokalne za njega
- autoescape - omogućava da se autoescaping aktivira i deaktivira

Autoescaping (1)

- Autoescaping nije uključen po defaultu, iako štiti od XSS napada
- Razlog je znatno veće vreme koje bi template engine utrošio na procesiranje
- Jinja2 je template engine opšte namene, koji se ne koristi samo za generisanje html-a i xml-a
- Ako je uključena autoescape ekstenzija, može se koristiti *autoescape* tag kojim se zadaje da li je autoescaping aktivan u datom bloku

Autoescaping (2)

```
{% autoescape true %}
    Autoescaping is active within this block
{% endautoescape %}

{% autoescape false %}
    Autoescaping is inactive within this block
{% endautoescape %}
```

High Level API

- Koristi se za učitavanje i renderovanje template-a
- Sa, druge strane, dostupan je i Low Level API, koji se koristi za pisanje ekstenzija
- Osnovne klase koje se koriste su:
  - Environment - osnovna komponenta
  - Template - kompajlirani template
  - Context - konteks template-a, čuva varijable
- U nastavku će biti dat kratak pregled najbitnijih osobina, dok je kompletna dokumentacija dostupna na linku

Environment

- Osnovna Jinja komponenta koja sadrži deljene varijable, poput konfiguracija, zatim filtere, testove, glabalne funkcije itd.
- Instance ove klase se mogu menjati, ali se to ne preporučuje ako se dele ili ako je neki template već učitan
- Omogućava podešavanje niza opcija, što se vrši pri inicijalizaciji objekta

Environment - inicijalizacioni parametri (1)

- Može se podesiti koji će string označavati početak bloka (što je po defaultu '{%'), koji će označavati kraj itd.
- Dalje imamo i pomnuta trim_blocks, lstrip_blocks i autoescape podešavanja, koja po defautlu imaju vrednost False.
- Može se specificirati koje će ekstenzije biti uključene
- Navodi se loader koji će se koristiti za dato okruženje

Environment - inicijalizacioni parametri (2)

- Primer:

```
env = Environment(lstrip_blocks = True, autoescape =
 True, extensions=['jinja2.ext.i18n'])
```

- Ako je template kreiran direktno preko Template konstruktora, okruženje se kreira automatski kao deljeno okruženje. To znači da više template-a mogu imati isto anonimno okruženje.
- Environment čuva i rečnike filtera, testova i globalnih varijabli

Kreiranje sopstvenih filtera i testova i dodavanje globalnih varijabli

- U rečnike filtera, testova i globalnih varijabli možemo dodati i sopstvene
- Uputstvo za pisanje i dodavanje filtera
- Uputstvo za pisanje i dodavanje testova
- Uputstvo za dodavanje globalnih varijabli

Učitavanje template-a (1)

- Za učitavanje template-a (sa fajl sistema ili iz nekog drugog resusra) koriste se loader klase
- Na raspolaganju je više ugrađenih loader-a, a moguće je i napisati sopstveni, koji tada treba da nasledi roditeljsku klasu svih loader-a - BaseLoader
- Za učitavanje template-a sa fajl sistema koristi se FileSystemLoader (navodi se putanja do template-a u vidu string argumenta)
- Za učitavanje iz Python paketa, koristi se PackageLoader

Učitavanje template-a (2)

- Loader se navodi pri kreiranju Environment objekta:

```
env = Environment(loader =
PackageLoader('jinjatest', 'templates'))
```

ili

```
env = Environment(loader =
FileSystemLoader("./jinjatest/templates"))
```

- U primeru jinjatest je paket unutar src paketa u kome se nalazi folder templates gde su template-i, a modul u kom se inicijalizuje loader je direktno u src paketu

Učitavanje template-a (3)

- Ako je konfigurisan loader, konkretan template se učitava pozivanjem get_template metode Environment klase
- Metoda prima:
  - naziv template-a (parametar *name*) - obavezno
  - *parent* na osnovu kog se formira puno ime template-a - opciono
  - rečnik koji sadrži varijable globalne za template (parametar *globals)* - opciono

Učitavanje template-a (4)

- Primer :

```
gvars = {"name" : "Pera", "surname" : "Peric"}
template = env.get_template('test', globals = gvars)
```

Renderovanje template-a (1)

- Po učitavanju template-a dobijamo Template objekat
- Renderovanje template-a vriši se pozivanjem *render* metode kojoj takođe možemo proslediti rečnik varijabla koje će biti dostupne u template-u
- Naredna dva rade isto:

```
template.render({"name" : "Pera", "surname" : "Peric"})

template.render(name = "Pera", surname = "Peric"))
```

Renderovanje template-a (2)

- Render metoda vraća renderovani template kao unicode string, koji dalje možemo ispisati, ili snimiti u fajl:

```
rendered = template.render(name = "Pera", surname =
"Peric")
with open("output/test.html", "w") as file:
    file.write(rendered)
```

Primer

- Primer učitavanja i renderovanja template-a sa snimanjem u fajlu:

```
env = Environment(trim_blocks = True, lstrip_blocks
= True, loader = PackageLoader("forms",
"templates"))

template = env.get_template("form.html")

t = template.render(form = model)

print(t)

with open("output/form.html", "w") as f:

    f.write(t)
```

Low level API

- API niskog nivoa daje mogućnost pristupa funkcionalnostima koje mogu biti korisne za razumevanje nekih implementacionih detalja, debagovanje ili razvoj ekstenzija
- Više o razvoju sopstvenih ekstenzija